1995-09

Integration of hypermedia capability into NPSNET IV.8, a large-scale real-time distributed simulation system

Shaffer, Alan Bruce.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

INTEGRATION OF HYPERMEDIA CAPABILITY INTO NPSNET IV.8, A LARGE-SCALE REAL-TIME DISTRIBUTED SIMULATION SYSTEM

by

Alan Bruce Shaffer

September 1995

Thesis Advisor: David R. Pratt
Co-Advisor: John S. Falby

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1995	Master's Thesis

4. TITLE AND SUBTITLE
INTEGRATION OF HYPERMEDIA CAPABILITY INTO NPSNET IV.8, A LARGE-SCALE REAL-TIME DISTRIBUTED SIMULATION SYSTEM

5. FUNDING NUMBERS

6. AUTHOR(S)
Shaffer, Alan Bruce

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

In today's large-scale computer simulations, the need has arisen to provide the user with an informed means of navigating a virtual world, such as with hypermedia. While previous work has been conducted in the area of hypermedia in a small-scale, single-user virtual world simulation, there has been no design and implementation of hypermedia capability into a large-scale, real-time networked simulation, such as the NPSNET project.

The method chosen for this thesis was to expand upon the Hyper-NPSNET project, a small-scale virtual world simulation system with hypermedia capability, to provide this capability to NPSNET. Access to video, audio, graphical, and textual data is provided via "anchors" placed throughout the world. Additionally, a GUI interface panel was developed which allows the user to navigate throughout the virtual world, and access information stored in an associated database. The interface panel allows the user to view specific information from anchors within the 3D world. Additional utility is provided for authoring of new anchors in the world.

The result of this thesis is that NPSNET now possesses a full hypermedia capability, controllable via a system interface panel. A better overall training environment is provided because users can now readily access database information while traversing this large-scale, multi-user virtual simulation.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Hypermedia, NPSNET, Hyper-NPSNET, Virtual Worlds	72
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited

INTEGRATION OF HYPERMEDIA CAPABILITY INTO NPSNET IV.8, A LARGE-SCALE REAL-TIME DISTRIBUTED SIMULATION SYSTEM

Alan Bruce Shaffer
Lieutenant, United States Navy
B.S., U.S. Naval Academy, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

September 1995

Author:

Alan Bruce Shaffer

Approved By:

David R. Pratt, Thesis Advisor

John S. Falby, Co-Advisor

Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

In today's large-scale computer simulations, the need has arisen to provide the user with an informed means of navigating a virtual world, such as with hypermedia. While previous work has been conducted in the area of hypermedia in a small-scale, single-user virtual world simulation, there has been no design and implementation of hypermedia capability into a large-scale, real-time networked simulation, such as the NPSNET project.

The method chosen for this thesis was to expand upon the Hyper-NPSNET project, a small-scale virtual world simulation system with hypermedia capability, to provide this capability to NPSNET. Access to video, audio, graphical, and textual data is provided via "anchors" placed throughout the world. Additionally, a GUI interface panel was developed which allows the user to navigate throughout the virtual world, and access information stored in an associated database. The interface panel allows the user to view specific information from anchors within the 3D world. Additional utility is provided for authoring of new anchors in the world.

The result of this thesis is that NPSNET now possesses a full hypermedia capability, controllable via a system interface panel. A better overall training environment is provided because users can now readily access database information while traversing this large-scale, multi-user virtual simulation.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

During the course of my thesis work, there were many people who were instrumental to me. Without their guidance, help, and patience, I would have never been able to accomplish what I have here. I would like to take this opportunity to acknowledge some of them.

My first thank you's must go to my thesis advisors, Assistant Professor David Pratt and John Falby. In the early stages of my thesis research when I really had no concept of how to take that "first step," they provided me the inspiration and a seemingly endless source of ideas on how to go about this huge undertaking. For their advice and guidance I am very grateful.

Dr. Michael Zyda, while not my thesis advisor, had as great an impact on my time here. As a classroom instructor, his infectious enthusiasm never ceased to carry over to the students, and as a leader within the NPSNET Research Group, it is he who ensures that only the best product is developed. He was instrumental in peaking my interest in the field of computer graphics, and in the larger world of education and learning.

They say no one can make it through this school without the help of his fellow students, and I found this to be absolutely true. My classmates in section CS-41 were the best, and supported one another both in and out of the classroom. I must, however, single out one individual who played a particularly important role in my time here. LT Fred Lentz began as a good lab partner, but eventually became a great friend. His sense of humor carried me through many of the tough times when it looked like this thesis would never be completed, and I thank him.

Finally, and most importantly, I must give immense thanks to my wife Sandra and our two children Ashley and Sam. Without them, I simply would not have been able to do any of this. Their love and support during those long nights of "cramming" before a test and being away at my second home in the computer lab were immeasurable. This thesis is as much a credit to them as it is to me.

I. INTRODUCTION

In today's world of computer simulation, the need to provide the user with an efficient, informed means of navigating cyberspace is of utmost importance. This fact is especially true when applied to virtual world applications in a training role, where computer resources and time are often at a premium. One extremely effective means to this end is through the application and use of Hypermedia capabilities in a virtual world environment. The focus of this thesis is to implement such a capability into a large-scale networked, distributed simulation system, specifically, the NPSNET research project being conducted at the Naval Postgraduate School (NPS) in Monterey, California [ZYDA94]. This research expands on previous work conducted in this area, including the recently developed Hyper-NPSNET, an offshoot of the NPSNET project, where multimedia is imbedded into a small-scale 3D virtual environment [LOMB93].

A. WHY HYPERMEDIA?

The use of hypermedia in computer systems allows large amounts of data of various formats to be managed in a relatively easy manner. The concept of hypermedia is simply an extension of hypertext, which has been in use for some time on many popular systems. An example of this is the Microsoft Windows help facility. This hypertext-based tool allows a user to sort through a vast quantity of textual assistance information by selecting a highlighted topic of interest in one area, or page, of the help system. This action immediately sends the user to information pertaining to the desired topic, negating the need to cycle through all intermediate, extraneous information. Without this hypertext capability, a user might well find that using such a help facility is just not worth the time needed to find the data he is looking for, not to mention the fact that it may be nearly impossible to do so in some cases, due to sheer quantity of data.

The concept of hypertext can be extended to include not only management of textual information, but management of other data as well, such as graphical images, sound, and video. This extension forms the foundation for a hypermedia-based system. Data such as graphical stills, audio, and video requires much more storage capacity than textual data, and the need to efficiently access and manipulate this vast information becomes all the more crucial in a real-time computer system such as NPSNET.

B. VIRTUAL WORLD ENVIRONMENTS

With the advent of high-scale real-time virtual reality (VR) systems, computers have become an excellent, often necessary, training tool. Application areas range from the military to the corporate world to formal educational institutes. Systems have been built which allow, for instance, a military trainee to be placed in an otherwise hazardous environment, such as on a battle field or within a burning space onboard a ship. With VR computer systems, the trainee is able to conduct the sort of real training that in the past might have required many, many more hours and dollars using more traditional training techniques, often placing the trainee in danger. In fact, these restrictions often made certain hazardous training prohibitive. With VR training systems, however, there are no such restrictions based on the hazards of a situation, and in fact, this type of training is tailor made for the virtual world. Of course, the need for more traditional training cannot be superseded completely, but VR systems can provide much of the early work, where people are more apt to make costly mistakes.

In distributed real-time VR systems, not only is individualized training possible, but scenarios can be developed which involve multiple users spread across a network, each participating from his own site, but all part of a common VR world. The advantages here include all of those common to computer virtual world systems, with the additional benefit that a given participant can receive direct human responses to his actions, as opposed to interacting with preprogrammed computer generated entities. While these latter entities are

valuable in broadening the scope of some scenarios, the benefits of having other actual humans acting in the virtual world is obvious.

To return to the earlier examples, hypermedia could enable a user, while involved in training scenario, to access important data concerning a particular piece of enemy artillery, or the routing of a fire main within the structure of the ship. In each of these cases, the ability to display graphical images and video would be a valuable augmentation to simple displaying of textual information. Would a soldier not gain much more by actually seeing an enemy tank performing some typical maneuver, as opposed to just reading a laundry list of its vital statistics and capabilities?

Beyond data access, hypermedia allows a trainee in such a scenario to "jump" around within the virtual world. In the ship example, after fighting a simulated fire, the trainee could immediately transport himself to, say, the engineering spaces to conduct reduction gear training. In this case, preset hypermedia links will have been established connecting different areas of the virtual world, which would allow the user to navigate in such a manner.

C. HYPERMEDIA IN NPSNET

NPSNET is a networked, distributed simulation system allowing multiple users to share the same virtual environment and interact with one another [ZYDA94]. It currently possesses no real hypermedia capability, although a descendant has been built, known as Hyper-NPSNET, whose objective is to provide a single-user limited hypermedia capability from within an NPSNET-like environment. Hyper-NPSNET will be discussed in greater detail in Chapter II. As mentioned previously, the objective here is to develop a hypermedia framework to be incorporated into the current version, NPSNET-IV.8.

To completely implement a full hypermedia system requires: establishing a multitude of links throughout a virtual world; definition of anchors which provide links to textual, graphical, video, and sound data; creation of a database from which the user can access information concerning any entity or object in the virtual world; and, providing an

"authoring" capability, which allows the user to create his own anchors within the world and add data into the database. To undertake this completely into a simulation system of the magnitude of NPSNET is a tall order, and far beyond the scope of this work. But, one can still establish an initial hypermedia capability by implementing some of the above, such as creating some major links within the virtual world, providing an initial database of information with associated anchors in the world from which to access data, and perhaps most important, providing an authoring capability allowing later additions to the database and anchor structure, thus allowing for the hypermedia system to grow as time passes. This latter implementation has been the thrust of this thesis work. To facilitate use of this system, a scripted scenario within NPSNET was developed, thus allowing full use of the hypermedia capabilities, but geared toward the database information and associated anchors built here.

D. CHAPTER SUMMARY

This thesis is organized in the following manner. Chapter II provides a brief background into the concept of hypermedia, and discusses previous work in the area of hypermedia systems. Chapter III outlines the basic hardware and software system requirements needed to use the NPSNET hypermedia system. Chapter IV presents the hypermedia framework developed here. This is followed in Chapter V by a look at the Interface-Control Panel, and how hypermedia functionality was incorporated into this. Also included here is a discussion of anchor authoring tools. Finally, Chapter VI includes conclusions and recommendations for further work in this area, followed by an appendix and reference list.

II. BACKGROUND AND PREVIOUS WORK

A. HYPERMEDIA BACKGROUND

Hypertext has been defined as "an approach to information management in which data is stored in a network of nodes connected by links. Nodes can contain text, graphics, audio, video as well as source code or other forms of data." [SMIT88] The concept of hypertext extended to include all forms of multimedia is called "hypermedia". The promise of hypermedia lies in its ability to produce large, complex, richly connected, and cross-referenced bodies of information that can be quickly and easily accessed.

The original idea behind hypertext was first put forth by Vannevar Bush, considered the grandfather of hypertext, in July 1945. He described a device called "memex" in which an "individual stores his books, records and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory." [BUSH45] Bush was concerned about the explosion of scientific literature at that time, and saw it as an impossibility to follow, even for specialists. The Memex system would store information on microfiche which would be kept on a user's desk. The user would access data through translucent screens, using a keyboard and a series of buttons and levers. He described an "associative indexing, the basic idea of which is a provision whereby any item may be caused at will to select immediately and automatically another. This is the essential feature of memex. The process of tying two items together is the important thing." [BUSH45] While the system was never implemented, its concepts laid the base for today's hypermedia systems.

In 1965, Ted Nelson first coined the word "hypertext" (non-linear text) and defined it as "a body of written or pictorial material interconnected in a complex way that could not be conveniently represented on paper. It may contain summaries or maps of its contents and their interrelations; it may contain annotations, additions and footnotes from scholars who

have examined it." [NELS65] Ted Nelson's dream since the early 1960s was to have all the world's literature available in one publicly accessible global online system. For thirty years, Nelson has worked on this vision of a "docuverse" (document universe) where "everything should be available to everybody." [NELS87] This work has produced Xanadu, a repository publishing system "intended to store a body of writings as an interconnected whole, with linkages, and to provide instantaneous access to any writings within that body." [NELS80] Xanadu has many interesting hypermedia concepts. For example, it overcomes the problem of generating unique names for new documents such that they can be found from any location on the network, and allows the ability to attribute royalties to the author of a work whenever it is retrieved across the network.

1. Nodes, Links and Anchors

A hypermedia system consists of nodes (concepts) and links (relationships). A node usually represents a single concept or idea. It can contain text, graphics, animation, audio, video, images or programs. It can be typed (such as detail, proposition, collection, summary, observation, issue) thereby carrying semantic information [RAO90]. Nodes are connected to other nodes by links. The node from which a link originates is called the reference and the node at which a link ends is called the referent. These nodes are also referred to as anchors. The contents of a node are displayed by activating links.

Links connect related concepts or nodes. They can be bidirectional thus facilitating backward traversals. Links can also be typed (such as specification link, elaboration link, membership link, opposition link and others) specifying the nature of a relationship [RAO90]. Links can be either referential (for cross-referencing purposes) or hierarchical (showing parent-child relationships). Figure 1 uses a graphical example to illustrate the relationships between nodes and links in a simple hypertext system.

Assuming that you start by reading the piece of text marked **A**. Instead of a single next place to go, this hypertext structure has three options for the reader: Go to **B**, **D**, or **E**. Assuming that you decide to go to **B**, you can then decide to go to **C** or to **E**, and from **E** you can go to **D**. Since it was also possible for you to go directly from **A** to **D**, this example shows that there may be several different paths that connect two elements in a hypertext structure.

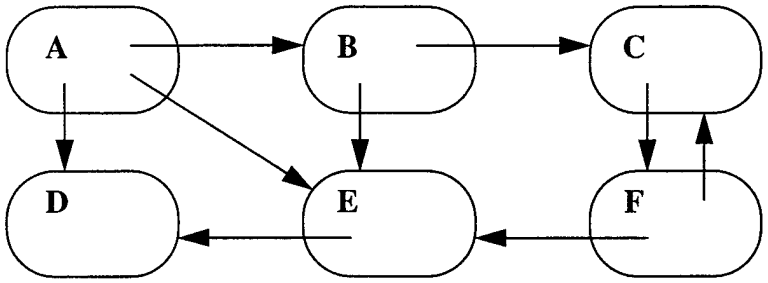

Figure 1:Simplified Hypertext Structure. From Ref. [NIEL90]

2. Basic Features of a Hypermedia System

A hypermedia system is generally comprised of several basic components. Though they are not all required in every case, the presence of each allows the user full flexibility within the system. These components are listed as follows:

1. A Graphical User Interface, with the help of browsers and overview diagrams, helps the user navigate through large amounts of information by activating links and reading the contents of nodes.

2. An authoring system with tools to create and manage multimedia nodes and links.

3. Traditional information retrieval (IR) mechanisms such as keyword searches, author searches etc. There are also attempts to incorporate structure queries along with content queries - retrieving a part of the hypertext network based on some user-specified criteria.

4. A hypermedia engine to manage information about nodes and links.

5. A storage system which can be a file system, a knowledge base, or a relational or object-oriented database management system.

3. Advantages of Hypermedia Format

Representing information in a hypermedia format has a number of distinct advantages. Hypermedia can support good browsing capability. It can provide the user better visual predominance of data, and more rapid navigation through huge numbers of entries. Also, hypermedia provides a dynamic nature to information, something that the printed word cannot readily provide, even in a traditional computer file structure. Finally, since electronic media can now store amounts of information which were once unimaginable, hypermedia in many cases provides the only means of accessing data in any manner, timely or otherwise. [COOK88][RAYM88]

B. PREVIOUS WORK

1. NPSNET

The Naval Postgraduate School Networked Vehicle Simulator IV (NPSNET-IV) is a low-cost, student written, real-time networked vehicle simulator that runs on commercial, off-the-shelf workstations (the Silicon Graphics IRIS family of computers) [ZYDA93]. The simulation reads and writes Distributed Interactive Simulation (DIS) 2.0.3 protocol data units (PDUs), and utilizes both SIMNET and MultiGen formatted terrain and model databases [PRAT94]. NPSNET is an ongoing project, used as a test bed for new areas of work within the NPSNET Research Group. Current areas of research include human insertion and articulation into the Virtual Environment, terrain database evaluation and improvement, integration of autonomous forces, and hypermedia integration, among others.

2. Hyper-NPSNET

Hyper-NPSNET is a real-time, single-user virtual environment providing the user with interactive hypermedia capability. Its main focus is to provide the underlying data structures to embed multimedia information in a real-time 3D virtual environment [LOMB93]. Unlike other similar projects, such as the Information Visualizer, Hyper-NPSNET allows the user to navigate throughout the virtual world unconstrained [CARD91]. In addition, it possesses the unique ability to attach up to four types of media information to a single location in the world: audio, video, graphical, and textual.

Hyper-NPSNET builds a chain of links for each anchor visited by the user, allowing him to "back out" in reverse order. At each anchor, the user has the ability to obtain information about that particular location in the virtual world, such as location name and world coordinates, or he/she may choose to playback audio, video, graphical, or textual information previously attached to that anchor [LOMB93]. Also, Hyper-NPSNET allows the user to create and define new anchors in the world, known as Authoring. These newly created anchors may be attached to either locations or actual entities, and can then have attached to them specific multimedia information for later retrieval.

A hypersystem is defined as the fundamental data structures that hold individual node information and all underlying links between anchors and information nodes. At the lowest level is the HyperNode, which is the basic information entity in the system. Above this, Anchors contain up to four HyperNodes, or links to graphical, video, audio, and textual information. The collection of all the Anchors in the database make up the HyperSystem. It is at this level that anchors are created, deleted, and modified. [SERB94]

3. ATOC^{3}IPT

The Automated Tactical Operations Command, Control, Communications, and Intelligence Planning Tool (ATOC^{3}IPT) was designed to aid commanders and their staff in the decision making process, as well as to provide tactical training. ATOC^{3}IPT is based on Hyper-NPSNET, but is designed to be used with multiple users, and with multiple overlays

providing anchor information in the virtual world. The hypersystem used in Hyper-NPSNET was extended for ATOC^{3}IPT to include multiple permission-protected overlays with multiple instances of each type of multimedia information available at any specific location [SERB94]. As well, ATOC^{3}IPT provides a highly enhanced graphical user interface (GUI) through which the user can access the overlay information.

4. NPSNET Control/Interface Panel

With each new addition to NPSNET, the multitude of key strokes and input devices used to control the system has become somewhat overwhelming. Since the focus of NPSNET research had always been more in line with implementation of new entities and features to explore "proof of concept," a well structured user interface had never been a driving factor. With the increasing functionality being built into NPSNET, however, it has become a requirement that some manner of interface panel be designed for a user to more readily utilize this ever growing system.

With this in mind, a fully functional control and interface panel was recently designed and constructed for NPSNET. Utilizing object-oriented techniques, an application framework was designed to enable the rapid creation and incorporation of interface modules. This framework provides the capability for reusable interface components that can be plugged into modules, while also offering the flexibility to customize these interface components, or build new ones as required. [MCMA94]

The original interface panel was developed using the Iris Viewkit system, a toolkit which allows for encapsulation of Motif widgets into C++ classes. The panel provides the user the capability to both control and get feedback from another networked workstation running NPSNET. Communication between the panel and the host computer running NPSNET is handled via a defined data structure used as a protocol to be passed over a broadcast socket stream. One drawback lies in the fact that the original socket protocol made no use of a host computer identity, so multiple sessions of NPSNET running on a local network will be controlled by the same interface panel, and the panel thus receive

conflicting feedback from the multiple hosts. Similarly, multiple interface panels running at the same time will update a single NPSNET session, and in the case where multiple interface panels and multiple sessions of NPSNET are all running on the same network port, the results are unpredictable.

III. HYPERMEDIA SYSTEM REQUIREMENTS

In order to have full use of the NPSNET Hypermedia System, there are a number of requirements one must first have. This section will outline these requirements, as well as some discussion on the software tools used to create the system.

A. SOFTWARE REQUIREMENTS

1. NPSNET-IV.8

First and foremost, the Hypermedia System and its associated interface are an integral part of the NPSNET project. The code for NPSNET version IV.8 includes the hypermedia facilities. Specifically, the bulk of the Interface Panel code is located under the NPSNET subdirectory:

```
src/interface/panel/
```

The main driver code and Makefile for the Interface Panel are located under the directory:

```
src/apps/panel/
```

Code contained within the main simulation loop of the NPSNET main() function relating to the Interface Panel is bracketed by the `REMOTE_PANEL` definition. Specifics of the modified code are described in a later section of this thesis.

2. ViewKit Software Development Toolkit

The ViewKit system, a part of the Silicon Graphics Irix 5.3 Operating system, is a C++ toolkit that makes applications development much easier. It provides a collection of high-level user interface components and other support facilities that must typically be implemented in every application, such as windows, menus, and dialogs [VIEW94]. To be able to run NPSNET with the full hypermedia system capability, one need only have the ViewKit eoe files loaded onto the system. However, a brief discussion on the various ViewKit tools used in developing this thesis are presented here for explanatory purposes.

ViewKit does not replace Motif for building menu and window widgets. In fact, it uses Motif widgets to implement all of its user interface components; also, one can directly call Motif functions to create and manipulate widgets in a ViewKit application. The ViewKit architecture helps mask much of the complexity of programming with Motif.

While ViewKit possesses a very extensive library of predefined classes, it can be used effectively with only a subset of these, which was the case for this thesis. The primary components used were *VkApp*, *VkComponent*, and *VkSimpleWindow*. *VkApp* is used to define an overall application for the program, and its member function *run()* is called once all other components have been established. The *VkApp* class handles application-level tasks such as Xt initialization, event handling, window management, cursor control, and application busy states. Figure 2 shows a code extract from the Interface Panel *main()* function which defines the *VkApp* and the primary *VkComponent* Panel.

```
...

// instantiate the viewkit classes
VkApp *app = new VkApp("Panel", &argc, argv);
Panel *panel = new Panel("panel");

// show the panel widget
panel->show();

// run the application
app->run();

...
```

Figure 2: Use of VkApp Class in the Interface Panel *main()* Function

In the above code extract, *app* is declared as the *VkApp* application variable. A Panel object pointer is declared, and then shown. The *show()* member function is, of course, not called until the entire Panel object has been built, which is the bulk of the program. Then finally, the application is run.

All ViewKit components are derived from the abstract base class *VkComponent*, which defines a basic structure and protocol for all components. The *VkComponent* class

sets up the basic component and callback structure of all its inherited classes. *VkComponent* can be used as an object class itself, but is most often used as the base class of other inherited ViewKit classes, e.g., the *VkPreferencesDialog*, the *VkSimpleWindow*, and the *VkMenu* classes, all used extensively in this thesis work.

The *VkPreferencesDialog* class provides the basic structure for a dialog window which can have selection toggles, text entry windows, and radio buttons built into it. Each of these is itself a lower level *VkComponent*, and the *VkPreferencesDialog* provides a easy way of placing these widgets in one window. Additionally, this dialog window automatically provides the push-button necessary for preference selection. This class was used for the hypermedia preferences window and anchor editor.

The *VkSimpleWindow*, and its direct child class *VkWindow*, are a basic type of widget which provide a simple frame into which almost anything can be placed. The only key difference between these two is the fact that the *VkWindow* provides facility for a menu bar, while *VkSimpleWindow* does not. The *VkSimpleWindow* was used in this thesis for display of the hypermedia database files, as well as for the anchor lister window.

Finally, the *VkMenu* class provides for creation of all Motif style menu functions, such as top level menu bars, pull-down menus, pop-up menus, and the like. The class allows for the definition of menu items both statically and dynamically, as well as providing the capability to sensitize and desensitize menu option selectors.

B. HARDWARE REQUIREMENTS

1. System Requirements

The Hypermedia System, as with NPSNET, is designed to run on the Silicon Graphics Inc. family of graphics workstations. Because NPSNET is a very graphically intensive program, it can require significant computing power. Compounding this is the fact that playback of many hypermedia data files, particularly video files, can be very costly too. Because of this, better overall performance is realized on the more advanced SGI systems. An example of such a system would be the Onyx Reality Engine 2, with four 100

MHz processors, 128 Mbytes of main memory, and an integral Ethernet controller. Realizing that all users may not have the assets to posses such state of the art computing power, this system will operate just fine, with some speed degradation, on a lower level SGI workstation.

In order to realize its full capabilities, the hypermedia system requires that a sound system be present on the local workstation. Additionally, to make use of the Interface Panel, complete with all of the hypermedia functionality, a dual-station setup is required. This configuration would have NPSNET running on one workstation, while the Interface Panel runs on a second, local workstation.

2. Disk Storage Capacity

By their very nature, hypermedia files tend to be quite large in size. As such, in order to realize maximum performance from the hypermedia system, data files should be maintained locally, so as to negate network transmission times. Some of the data files used in development of this system range anywhere from 500Kbyte graphical image files to upwards of 20Mbyte video files. It is relatively easy to see that a hypermedia system of even moderate size will require a rather large hard disk storage capacity, on the order of Gigabytes.

3. Input Devices

At a minimum, the only input device required to use NPSNET with the hypermedia system is a mouse-type point-and-click device. This will provide the user the ability to control the vehicle in NPSNET using the Interface Panel controls and access all hypermedia functionality. A better system capability, however, would allow the user to switch between a mouse for hypermedia functions, and more realistic controls for the vehicle, such as a joystick and throttle setup. Future developments in the hypermedia area might also include a voice input device, e.g., a microphone, to allow access to database files, thus negating the need to have the user's hands tied up accessing menu controls.

IV. HYPERMEDIA FRAMEWORK IN NPSNET

A. HYPERMEDIA DATA STRUCTURES

The hypermedia framework used for NPSNET IV.8 derives from that of the Hyper-NPSNET system. Hyper-NPSNET is implemented as a series of C++ classes used to represent the various levels of the hypermedia system, or HyperSystem. This HyperSystem is designed as the fundamental data structure used to hold individual node information and the underlying links between anchors and nodes. Each of the components of the HyperSystem is described in detail below.

1. HyperNode

The HyperNode is the basic informational entity of the HyperSystem. It is implemented as a record structure holding its identification, type, and associated data file name. A node type can be one of four types: AUDIO, VIDEO, GRAPHIC, or TEXT, representing the four types of hypermedia information which can be stored at a particular location in the virtual world. For each node, the file name contains the path to the associated data file related to the node's type.

2. Anchor

The Anchor represents a particular location in the virtual world to which database information can be linked, or from where one can travel directly to another location in the HyperSystem. An anchor is essentially a container object that brings together HyperNode information into a manageable structure. The information associated with a particular anchor can be any or all of the aforementioned node types.

The Anchor is implemented as a C++ class made up of anchor identification, type, name, location, orientation, and attached node types, as well as functions to set and retrieve these values. Each anchor is identified internally by its unique identifier. The anchor name

is used only as a convenience to the user. The list of available anchor names can be displayed by the user by simple selection of a button in the hypermedia control section of the interface panel, to be described later. From this panel, the user can readily perform a jump to a particular anchor, which represents a location and orientation in the virtual world. The Anchor object stores this location, and the orientation is used to direct the user to a specific view point once he/she has reached the anchor position.

Currently, only TERRAIN anchor types are implemented. These anchors represent fixed locations in the virtual environment. Other anchor types, however, could conceivably be implemented. Examples include vehicle or entity-specific anchors which are attached to entities in the virtual world, or perhaps temporal anchors which are time constrained. These new anchor types would be dynamic, either changing location over time, or having only a limited lifetime.

3. HyperSystem

The HyperSystem provides the backbone of the overall hypermedia internal structure. It organizes the Anchors and HyperNodes in such a way that they can easily be accessed, created, modified, and deleted.

The anchors are organized into an array table structure, called the **AnchorTable**. This table is built as an array of C++ structures, where each structure contains an anchor index and a pointer to the associated anchor object. The array is initially instantiated to be of a maximum size, in this case 30 elements, and the **HyperSystem** always maintains a counter of the total anchors. In this way, a dynamic number of anchors can exist in a given world database, as long as the maximum is not exceeded. If it were desired, the allowable maximum could be increased as appropriate for the application (this is not currently a user selectable option), or a dynamic array type of structure could be alternately used. At any rate, by using an array data structure to organize the anchors in the hypermedia world, direct and immediate indexing of anchors is allowed, thus speeding up user access to an anchor, as for a jump or anchor modification.

The HyperNode objects are organized and stored in a similar manner to the Anchor objects. For an anchor table holding **n** anchors, there are necessarily **4n** nodes in the node table. Each Anchor object contains an index to its four possible information nodes, such that Anchor index 1 has attached to it HyperNode indexes 1 through 4, for example.

The Hyper-NPSNET system had previously allowed for anchor additions to the world database, but had no provision for deletion of anchors. Obviously, as a world database grows with time, certain anchors will become obsolete. The only means of cleaning out the database previously was through direct editing of the data file. This was deemed unacceptable, so an anchor deletion capability was added to the new system.

The anchor deletion capability posed an interesting set of problems, due to the nature of the anchor and node data structures being so closely tied to one another. First, when the user selects an anchor to be deleted, its index is accessed from the anchor table and the associated anchor object deleted. Then, to maintain the anchor table ordering, the indices of all anchors after the deleted one must be decremented. As well, the associated anchor table element pointers must be reset for these anchors. For example, suppose anchor **i** is deleted. Anchors **i+1** through **n** in the anchor table must now all have their indices decremented, and the anchor object pointer for the new anchor **i** would now point to the previous anchor **i+1**. This process must then be carried out for the remaining anchors through anchor **n**.

The interesting part comes when one considers the consequences of an anchor deletion on the node table. Since one anchor links to four nodes, the above process of decrementing table indices and resetting pointers must be carried out in the node table by groups of four. To illustrate, in the above example where anchor **i** was deleted, the associated nodes to delete would be indices **4i**, **4i+1**, **4i+2**, and **4i+3**. With this in mind, node **4(i+1)** would now become the new node **4i**, node **4(i+1)+1** would become **4i+1**, and so on. At the same time, the pointers would need to be reset such that the new node with index **4i** has a pointer to the previous node **4(i+1)**.

In deleting anchors from the world database, it was important that two things occur. First, the deleted anchor needed to be reflected on the interface panel **AnchorLister** in real time. In other words, the user needed to see the anchor disappear from the Lister, as well as in the virtual world. Also, the deletion needed to be maintained so that later, when the world was saved back to the database, the new information was stored correctly. By using the above described data structures, both of these goals were met.

B. COMMUNICATIONS PROTOCOL

The communications mechanisms developed for the original Interface Panel were grouped into three categories: communications between Panel classes, communications between NPSNET and the Panel, and Panel communications with other DIS network traffic. For this thesis work, two of these categories were updated: Panel class intercommunication, and the communication between the Panel and NPSNET. Figure 3 shows an overview of the updated communications structure with major components of the hypermedia system enclosed by a dashed box.

1. Panel Class Intercommunication

Updating the Panel intercommunications structure involved combining the individual Hyper-NPSNET and Control-Interface Panel communications structures. Previously, the interface panel structure provided communications between the vehicle classes and the window ViewKit widgets in order to display vehicle specific controls. In designing the new hypermedia system, these communications needed to be maintained, while also incorporating the necessary HyperSystem communications.

To provide the hypermedia functionality needed in the interface panel, a HyperButtons class was added to the interface panel. This class provides ViewKit button widgets for selecting various information node data, jumping back through the anchor stack, and displaying the list of available anchors (details of these functions are discussed in Sections V.C and V.E of this thesis). In each of these cases, selection of a button sends a message to the Panel class, which then performs the appropriate action. For example,

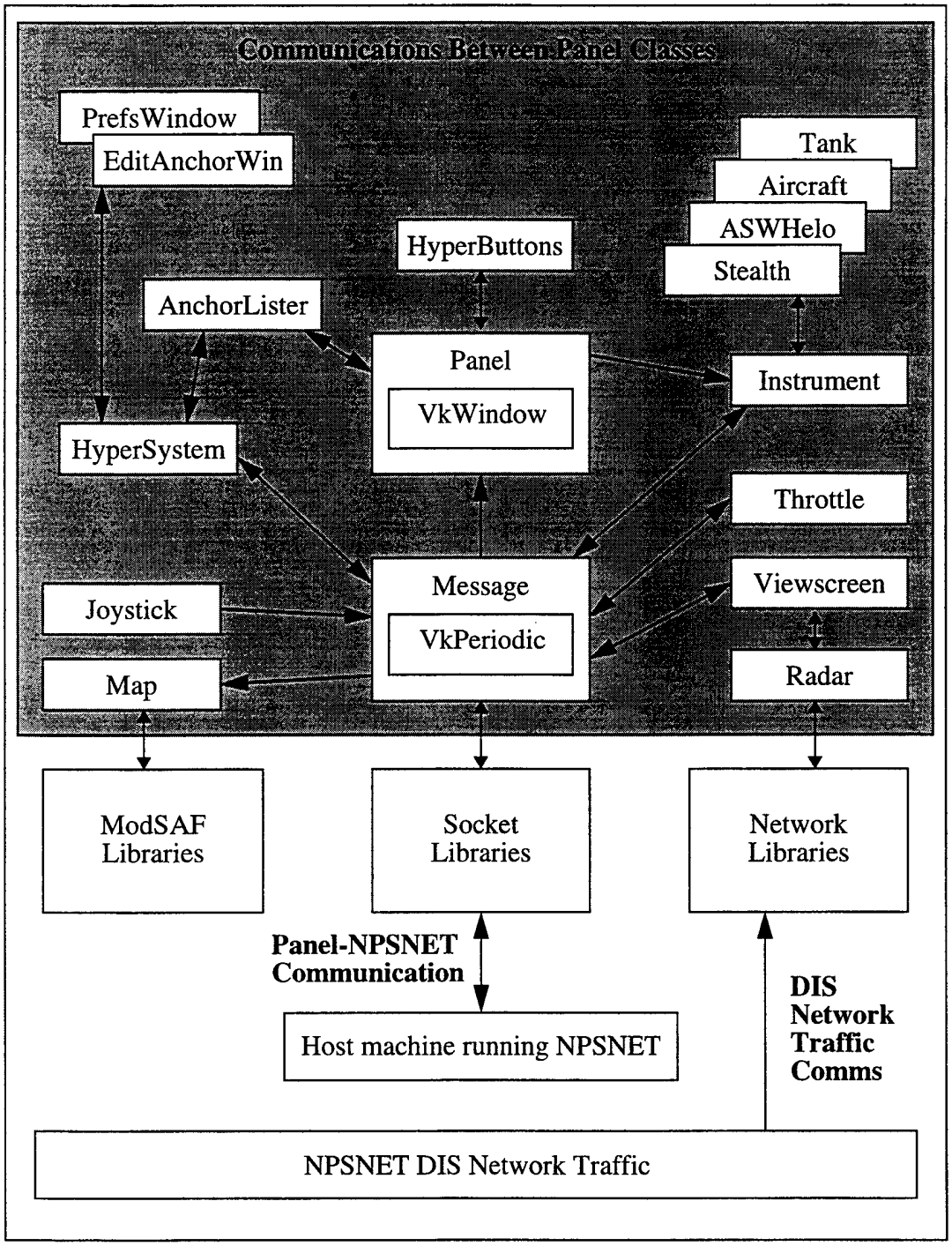

Figure 3: Communications within the Interface Panel

when a hypermedia world is actively open, and an anchor current, selection of the **Video** button in the HyperButtons class will cause the Panel to bring up the *movieplayer* utility with the associated video data file.

Similarly, selection of the HyperButtons **Anchors Available** button widget will send a message to the Panel to display the AnchorLister object window. Subsequent interaction with the Anchor Lister will then send appropriate messages back to the Panel and to the HyperSystem. For example, selection of the Jump button on the Anchor Lister will send a message to the Hypersystem to get the anchor's position, which will in turn write a PDU socket to NPSNET to update the virtual world.

2. Panel-NPSNET Communication

a. Hypermedia Socket Structure

Communication between the Interface Panel and NPSNET is handled through a PDU communications socket. Previously, a socket structure was defined as seen in Figure 4. This maintained all the necessary vehicle data for display on the panel, as well as for entity updating within NPSNET. This socket structure holds the principle changes made to the Panel and NPSNET intercommunications. Note that the socket in the figure includes an element called **hypermediaData** of type **HYPER_DATA**. It is this element that has been added to the structure, and which maintains the pertinent hypermedia information to be passed between the panel and NPSNET.

Figure 5 shows the new structure added for the hypermedia data communications. The structure, called **HYPER_DATA**, was designed to allow for complete control of the NPSNET anchor system, while still keeping the data packet size to a minimum to maintain network speed. The first element, **anchorArray**, is used to store the location and orientation data for each anchor in the HyperSystem. Whenever a modification is made to any of the anchors in the HyperSystem, the **anchorArray** information is updated, and a PDU socket written to NPSNET. This ensures that any anchor changes are immediately registered visually in the virtual world.

```
typedef struct {

    // Identify the type of data structure being passed
    double status;
    unsigned short type;
    unsigned short length;

    // Throttle data required to read the throttle position
    //   (-1.0 to 1.0) and to set the throttle input with the
    //   scale widget
    float throttleSetting;

    // Joystick data required to read the joystick position
    //   and set input as required (each is between -1.0 and 1.0)
    float joystickX;
    float joystickY;

    // Vehicle settings read from NPSNET only
    float positionX;
    float positionY;
    float positionZ;

    float altitude;
    float heading;
    float pitch;
    float roll;

    float velocity;
    float gunAzimuth;
    float gunElevation;

    EntityID vehicleID;

    // Settings sent to execute NPSNET actions
    EntityID targetVehicleID;
    HYPER_DATA hypermediaData;

    BYTE attachMode;  // Including tether, attach, target,teleport

    BYTE weaponsMode; // Including primary, secondary,tertiary,
                      //   targetingEnable

    BYTE hudMode;           // Including hudEnable

    BYTE environmentMode;  // Including fogEnable, wireframeEnable,
                           //   textureEnable, cameraEnable

    // Variable settings to control NPSNET functions
    BYTE fogSetting;
    BYTE hudSetting;

} GUI_MSG_DATA;
```

Figure 4: Socket Communications Structure. From Ref. [MCMA94]

```
typedef struct {
   float positionX;
   float positionY;
   float positionZ;
   float orientation;
} AnchorData;



typedef struct {

   // The array of anchors to be displayed in the virtual world.
   AnchorData anchorArray[MAX_ANCHORS];

   // Indicate whether a hypermedia file is currently open
   unsigned fileOpened;

   // Indicate whether or not anchors are to be displayed, and
   //   the count of anchors to be displayed in the panel.
   unsigned char displayAnchors;
   int anchorCount;

   // Indicate whether an anchor jump has been made on the panel.
   unsigned char jumpToAnchor;
   int jumpToThisAnchor;

   // Indicate whether the anchor system has been changed, so
   //   that NPSNET can redraw the anchors
   unsigned anchorsChanged;

} HYPER_DATA;
```

Figure 5: Hypermedia Communications Structure

The **fileOpened** element stores whether or not a hypermedia file is currently open at the interface panel. This is used to indicate to the NPSNET main() function simulation loop whether or not to process interface panel input data. The **displayAnchors** and **anchorCount** elements are used to tell the Anchor class whether or not it should add the anchor model to each of the active anchor array elements for display. This procedure is detailed in a later section.

The elements **jumpToAnchor** and **jumpToThisAnchor** are used to indicate to NPSNET that an anchor traversal has been performed at the interface panel. The former of these acts as the jump flag, and is set by the panel as part of the *Jump* button

callback. After a socket is written to NPSNET, this flag value is immediately set back to false. The latter element stores which anchor is being jumped to. It always stores the value of the last jump anchor, and is valid only if **jumpToAnchor** is true.

Because of the relative static nature of an anchor once it has been placed in the virtual world, there is no real need to redraw the anchor system in NPSNET with each new frame, or even with each PDU socket written from the interface panel. In fact, the anchor system needs to be redrawn only when an anchor has been modified in some way or when initially drawing it for the first time. This essentially describes the purpose of the last element in the **HYPER_DATA** structure, **anchorsChanged**. Whenever the HyperSystem has been modified in the interface panel, the PDU socket is modified so that **anchorsChanged** is set to true. When the next socket is written to NPSNET, **anchorsChanged** indicates that the anchor array must be rebuilt and the anchors redrawn. This procedure is handled by the Anchor class function *draw_anchors()*, discussed in the next section.

b. NPSNET Host Filter

The previous version of the Control-Interface Panel had included no capability to designate a specific NPSNET host workstation for which to provide interface. The Message class *read_socket()* function simply read in any and every socket put onto the net, and interpreted it accordingly. This situation worked fine for the case where only one Interface Panel and one NPSNET session were running simultaneously. The Panel would receive PDU sockets from only the single NPSNET session, and would interface correctly. Even if one desired a situation whereby two or more interface panels controlled a single session of NPSNET, such as in a tank crew training scenario, this case also performed correctly. This was because, again, only the one NPSNET session was sending feedback to the multiple panels.

The problem arose when one desired to run a multiple Interface Panel, multiple NPSNET session scenario. In this case, PDU sockets from every panel where

being read and acted upon by every NPSNET host, thus causing chaos. This occurred because the Message class made no use of a socket identification mechanism.

Looking at Figure 4, the socket structure has a data member called **vehicleID** of type **EntityID**. The **EntityID** type maintains a unique host, site, and entity identification number for each vehicle. This data, however, was not being used to differentiate among the different NPSNET sessions hosts when a particular panel read a PDU socket. The new interface panel, as well as adding the hypermedia capability, has also added a mechanism by which the user can designate specifically which local NPSNET host he/she wishes to control when running the panel. The Message class then checks the host and site identity for an incoming PDU socket, and then discards those that do not apply to the desired host.

With this added feature, users now have the capability to run all possible Panel-NPSNET combinations. This allows for the possibility of crew coordination training whereby multiple players controlling a single vehicle entity in the virtual world can fight a simulated exercise against other entities also being controlled by multiple players.

C. NPSNET ANCHOR CLASS

In order to provide the user an indication of an anchor location in the virtual world, it was necessary to build some type of visual marker into NPSNET. To do this, a new C++ class was defined in NPSNET to provide the required functionality. Also, an actual anchor model needed to be built for the display itself.

Figure 6 is a code extract from the NPSNET *anchor.h* file, showing the definition for the Anchors class. The class is made up principally of an array called **L_anchor**, which can hold a maximum number of anchors. Each anchor is represented by a locally defined structure, ANCHORNODETYPE, made up of three elements: a Performer dynamic coordinate system, or pfDCS, node; the anchor type; and, a flag showing whether or not the anchor is active in the system. Currently, an anchor is designed to be of only one possible

type: TERRAIN. The other component of the Anchors class is a Performer pfGroup node called **anchor**.

```
//local data structure used to hold the anchors
typedef struct {
   pfDCS *dcs;
   unsigned type;
   int active;
} ANCHORNODETYPE;


class Anchors {
public:
   Anchors();
   ~Anchors();

   void display_anchors();
   void dump_anchors();

protected:
   pfGroup *anchor;
   ANCHORNODETYPE L_anchor[MAX_ANCHORS];
   void makeanchor(int, pfVec3);

};
```

Figure 6: NPSNET *Anchors* Class Definition

The Anchors class provides a number of member functions, both public and protected. To begin, the class constructor performs two principle functions. It first loads the MultiGen flight file model for the anchor into the pfGroup node **anchor**. This provides the model to be rendered in the virtual world scene. Then, it initializes the **L_anchor** array by creating a new Performer dcs for each anchor, setting the anchor **type** to TERRAIN, and setting the **active** flag to false. This basically establishes the array of anchors, but without actual anchor models attached to any of the elements.

The next function, *display_anchors()*, is called in the NPSNET *main()* simulation loop. The function *display_anchors()* cycles through the array of anchors in the HyperSystem, sent over in the interface panel communications socket, and for each anchor location, makes an anchor. This is done by making a call to the Anchors protected member

function, *makeanchors*. The *makeanchors* function adds an **anchor** model to the current **L_anchor** array pfDCS node using a *pfAddChild* call, and sets the anchor's **active** flag to true. Once this process has completed for all of the anchors in the HyperSystem, the **L_anchor** array will be loaded with an **anchor** model attached to each of the appropriate dcs nodes. When the next scene is drawn in the *main()* loop of NPSNET, the Performer subtree representing the anchor system will render all of the current anchors sent from the interface panel. Figure 7 depicts the anchor model rendered in an NPSNET scene. The anchor in this figure has been attached to a building in the virtual world, and is drawn in such a place that it can be seen from a distance.

Finally, the Anchor class *dump_anchors()* function clears out all of the anchors in the Performer subtree. To do this, the anchor model for each L_anchor array index is removed from the appropriate pfDCS node using the pfRemoveChild call. Also, the **active** flag is set back to false for each element of the array. This function is called whenever Anchors Off is selected on the interface panel, or when the HyperSystem is terminated, i.e., the current hypermedia file is closed.

D. SUMMARY

The framework structure for the new NPSNET Hypermedia System was designed with the older Hyper-NPSNET structure in mind, but was expanded significantly beyond it. The interface was redesigned using the SGI ViewKit Toolkit, and incorporated into the NPSNET Interface Panel. Functionality was added to the hypermedia controls, such as the ability to not only add and modify anchors in a hypermedia database, but also to delete them, and the ability to inspect the stack of previous anchors visited and traverse directly to an earlier anchor.

Additionally, logic was added to allow the user to designate specifically which NPSNET host he/she was controlling from the interface panel. This functionality is essential to providing crew coordination training, whereby multiple users can control the

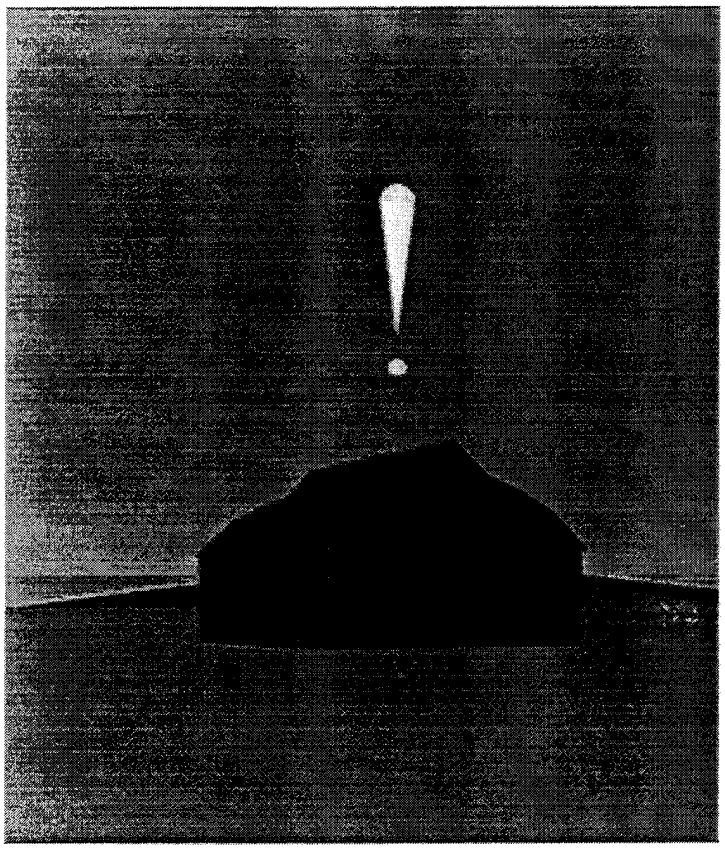

Figure 7: NPSNET Anchor Model Depiction

same vehicle entity in a virtual exercise. Alternately, an exercise can have multiple interface panels each controlling a different entity in the virtual environment.

V. HYPERMEDIA INTERFACE PANEL

The NPSNET Control-Interface Panel was modified in a number of ways to incorporate hypermedia functionality for the new NPSNET Hypermedia System. Menus were built to manipulate hypermedia database files, display anchors in the virtual world, traverse the world using these anchors, and access the information nodes while in NPSNET. In each case, the functionality mentioned above was developed using the SGI ViewKit Toolkit, and is detailed in this chapter.

A. HYPERMEDIA FILE ACCESS

The hypermedia system designed for NPSNET provides a user-friendly interface for interaction with hypermedia files stored on the user's local file system. Utilities are provided to open an existing hypermedia file, save the current anchor system to an existing file, or to save it under a new file name. Additionally, protection is built in so that a user cannot open a new file, close a file, or quit the system altogether without first being given the opportunity to save the current anchor structure. Figure 8 shows a rendering of the Interface Panel with major components, including the Hypermedia Controls, annotated.

File access is provided via the **Hypermedia** pulldown menu at the top of the interface control panel. Under this menu are several submenus, which can be seen in Figure 8. One of these submenus is titled **File**, and it is here that hypermedia files can be opened, saved, and closed.

1. Opening a File

There are two methods by which a user can designate a hypermedia file as the current one being used for an NPSNET session. Either an existing file can be opened, or a new file designated. The latter case is straightforward, and involves simply selecting **New** under the **Hypermedia-File** submenu. This will activate the various anchor capabilities

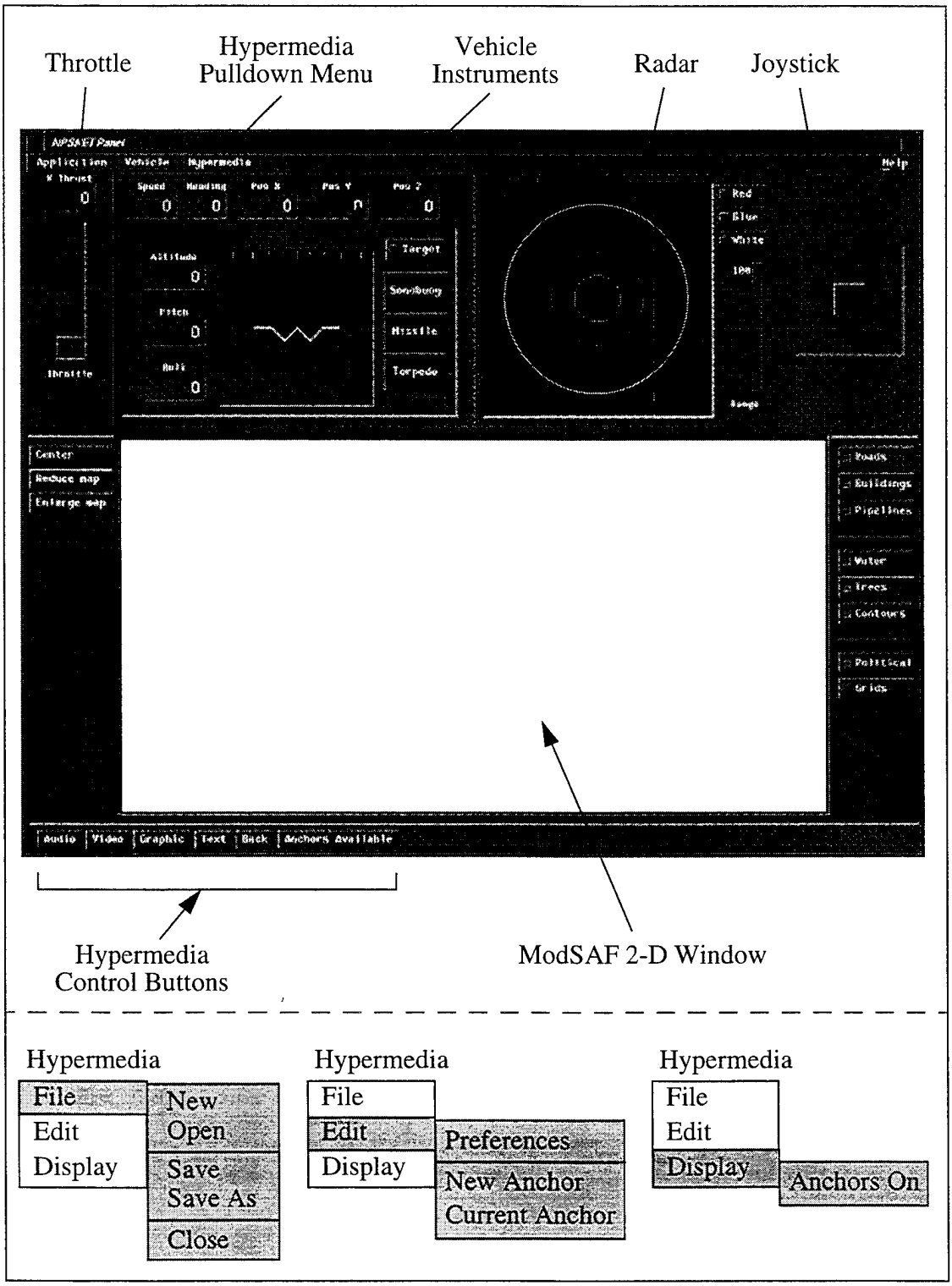

Figure 8: NPSNET Interface Panel and Hypermedia Menus

and functions. At this time, no anchors will be displayed in NPSNET, but the user can begin the process of building anchors and designating information nodes for the anchors.

The user can also open an existing hypermedia file and thus designate it as the current file in the NPSNET session. As with a new file, an existing database file can be opened under the **Hypermedia-File** submenu by using the **Open** selection. This button will display a file dialog box, as seen in Figure 9 below.

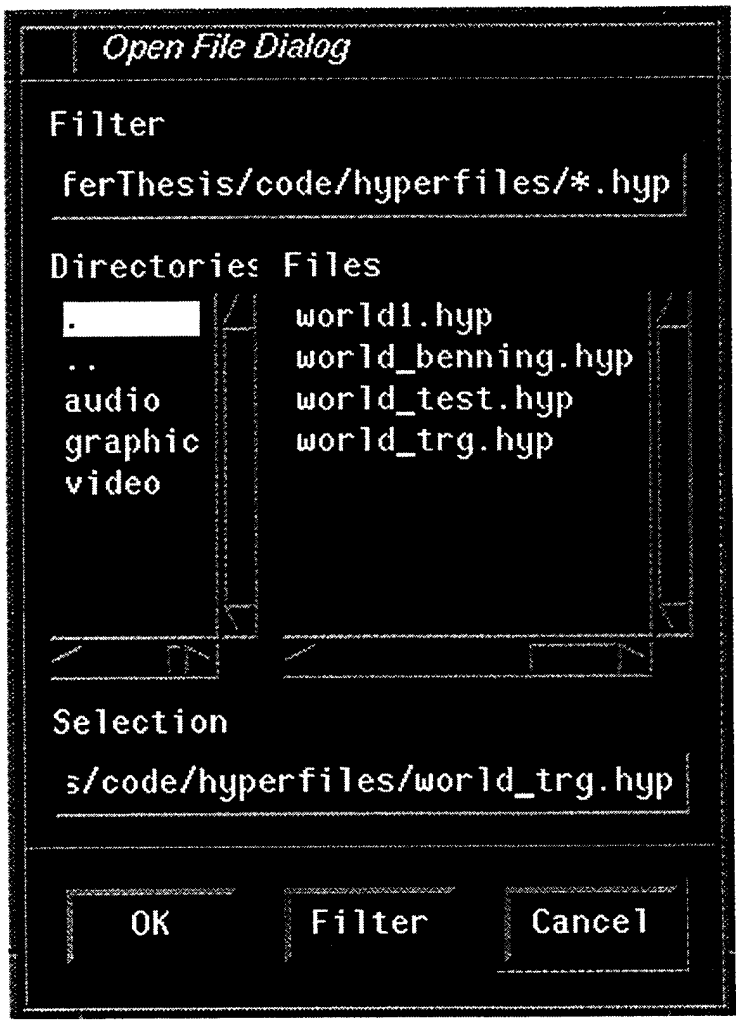

Figure 9: Interface Panel *Open File* Dialog Box

The file dialog box was built using the ViewKit *VkFileSelectionDialog* class. This class allows an initial directory path filter and file name to be specified, so that upon display of the dialog box these defaults are automatically entered into the appropriate windows. Within the dialog box, the user can then adjust the path filter to look for specific file types in a particular directory, and the file list window will automatically update. Once the desired file appears in the list window, the user simply selects this file and then selects the *Ok* button, which sets about a chain of events to open this file and read it into the HyperSystem data structure. The system's anchors, if any, will be displayed within the NPSNET virtual world. Alternately, the user can simply type the desired file path and name into the Selection text input window, followed by the *Ok* button.

2. Saving a File

As with opening a hypermedia database file, there are two methods to use in order to save the current virtual world anchor system to a database file. The first applies in cases when an existing file is currently opened, while the second applies when an anchor system is to be saved under a new file name. The first case is the simple one where an existing database file has been previously opened, and is accomplished by selecting the **Save** button under the **Hypermedia-File** submenu. This action will automatically update the currently opened database file to reflect whatever changes have been made to the HyperSystem, e.g., any anchors added, deleted, or modified, or information nodes edited, will be saved to the current file. The **Save** option is only made available in cases when an existing file has been opened, and will not be enabled when working with an original New file.

The **Save As** option under the **Hypermedia-File** submenu is used when the current anchor system is to be saved to either a new database file, or a designated existing file. Figure 10 below shows an example of the *Save File As* dialog box used to designate the file name to save a hypermedia system under. This dialog was developed using the ViewKit *VkFileSelectionDialog* class and, like the Open File dialog, has the ability to be displayed with default values for directory path and file name. The path filter can be modified to

reflect where the database file is to reside, and then the new file name typed is into the Selection text input window. The default file name *world_new.hyp* is used simply to provide the user a possible naming convention, and must be overwritten by a new name each time the function is used.

Figure 10: Interface Panel *Save File As* Dialog Box

3. Closing a File

To close a current hypermedia file, one need only select the **Close** option under the **Hypermedia-File** submenu. If the anchor system has not been changed at all, or since the last Save, the file will simply be closed and all pertinent anchor functionality will be deactivated. If, however, the system has been modified at all, then a dialog box will appear asking the user if he/she desires to save the current information to a database file. This dialog can be seen below in Figure 11.

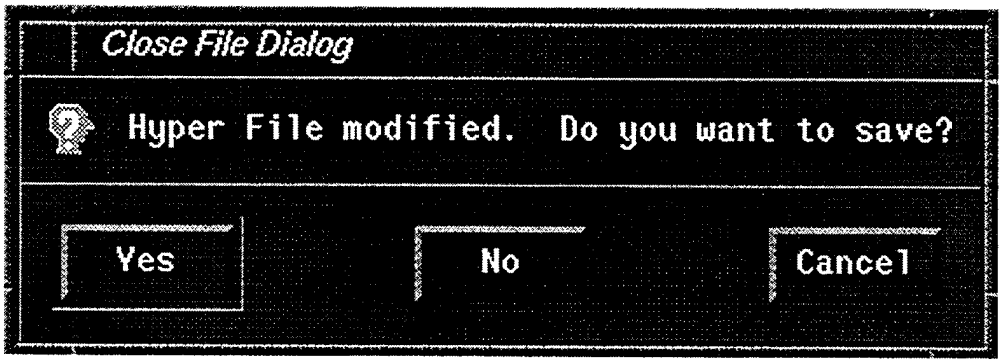

Figure 11: Interface Panel *Close* Dialog Box

Selection of the *Yes* button will either save and close the file, or bring up the *Save File As* dialog box, as appropriate. The *No* button will close the file without saving any modified information, while the *Cancel* button will terminate the close command and the current file will remain opened and unchanged.

B. HYPERMEDIA PREFERENCES

Once a database file has been opened and installed into the HyperSystem, several hypermedia system preferences become available for selection through the Control-Interface Panel. These preferences are detailed in this section.

1. Auto Anchor View

The Hypermedia Preferences dialog box can be displayed by selecting the **Preferences** option under the **Hypermedia-Edit** submenu on the Interface Panel. This

dialog box allows the user to select the *Auto Anchor View* function, as well as which types of information will be displayed. Figure 12 shows an example of the Hypermedia Preferences dialog box.

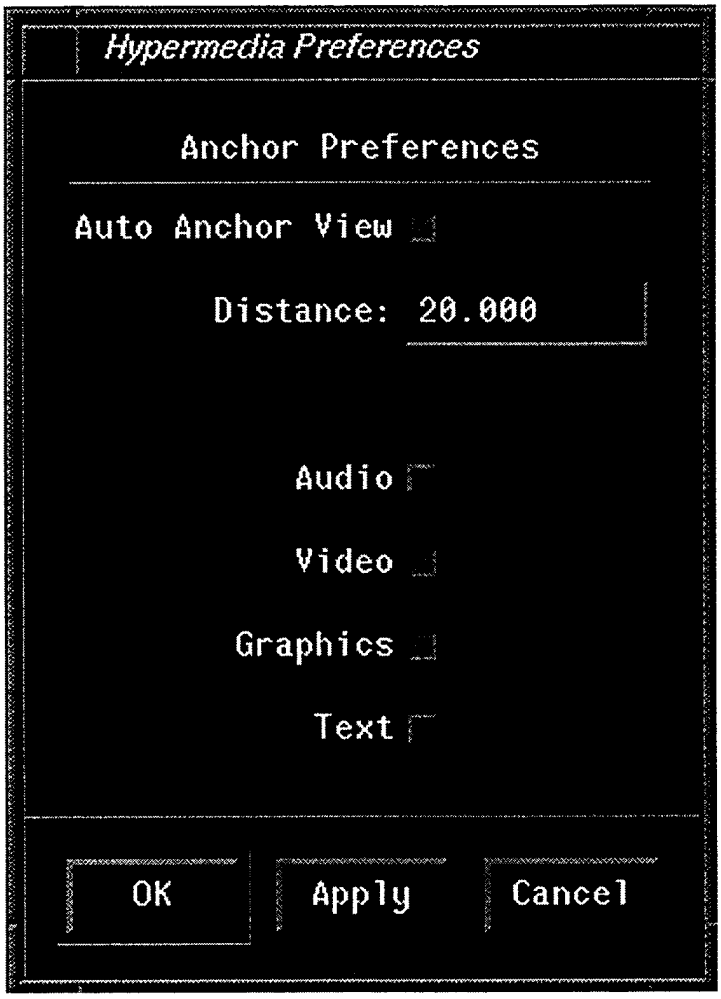

Figure 12: Interface Panel Preferences Dialog Box

Once the desired toggles are depressed, selection of the *Ok* button will apply the information and hide the dialog box. The *Apply* button will apply the selections but leave the dialog box up for further modification. Lastly, the *Cancel* button will cancel the dialog box and register no changes to the current HyperSystem.

Essentially, the *Auto Anchor View* function causes an anchor's information nodes to be automatically displayed when the vehicle being controlled comes within a specified range of that anchor. Which nodes are displayed is contingent upon selection of the *Audio*, *Video*, *Graphic*, and *Text* toggle buttons on the Hypermedia Preferences dialog, as well as whether the anchor in question has a link to a specific type of information node. The range out to which Auto Anchor View is enabled can be set in the Distance text input window and is initially set to a default value of 20.0 units.

The entered range value, in conjunction with the fact that Auto Anchor View is selected, will invoke a member function of the Message class to check the lateral distance between the vehicle in the virtual world and each anchor in the HyperSystem, with each socket written from NPSNET to the Panel. When the vehicle position, as read from the communications socket, shows it to be within the specified range of one or more anchors, the informations nodes of the closest anchor will display as windows on top of the Panel. The user can then remove these windows when he/she is finished viewing the data.

2. Anchors On/Off Toggle

The user also has the option of whether or not to display the anchors of a HyperSystem. When a file is initially opened, the default is for anchors to be displayed in the virtual world, but this option can be deselected by clicking the **Anchors On** option on the **Display** submenu under the **Hypermedia** pulldown menu. This action will toggle the button to read **Anchors Off**. Subsequent selection will toggle the option back to **Anchors On**, and so on.

By selecting anchors off, a flag variable in the communications socket between the panel and NPSNET is set to indicate that this option has been selected. This causes the Performer code in NPSNET to delete the anchor models from the anchor subtree, thus not displaying them in the virtual world. It is important to note that this is the only action undertaken by selection of the **Anchors Off** option, and that all other hypermedia

capability still exists between the HyperSystem and the NPSNET session, including anchor authoring and navigation, both discussed later.

C. VIRTUAL WORLD TRAVERSAL VIA ANCHOR NAVIGATION

In addition to the vehicle control capabilities provided by the Interface Panel via the graphical throttle and joystick, the Hypermedia System also allows vehicle movement throughout the virtual world using the anchor system. The various methods of performing such traversals are described here.

1. Anchors Available Window

As soon as a hypermedia file has been opened, or a new system initiated, the **Anchors Available** button along the lower edge of the Interface Panel will become enabled. Selection of this button will bring up the Anchors Available window within the Panel, as can be seen in Figure 13.

The Anchors Available window is built using the ViewKit *VkSimpleWindow* class, which essentially allows one to bring up a window in which to display anything. This window is broken down into two primary sections: the Anchor Lister and the Function Buttons. The Anchor Lister is a Motif *xmScrolledWindowWidgetClass* widget into which all the anchors of the HyperSystem are listed. This widget allows for selection of its listed items, which are returned to the callback function of the appropriate Function Button.

2. Jump Button

The Function Buttons allow for navigation between the anchors in a number of ways, and for deletion of anchors from the HyperSystem, and thus the list. The **Jump** button is used to traverse directly to the location of the selected anchor from the list. The **Jump** button will send the position of the anchor to NPSNET in the next outgoing communications socket, along with an indication that a jump has been made. NPSNET will then update the vehicle position and view orientation to that of the anchor. Also, due to the fact that a moving vehicle can make an anchor jump, the vehicle's speed and throttle

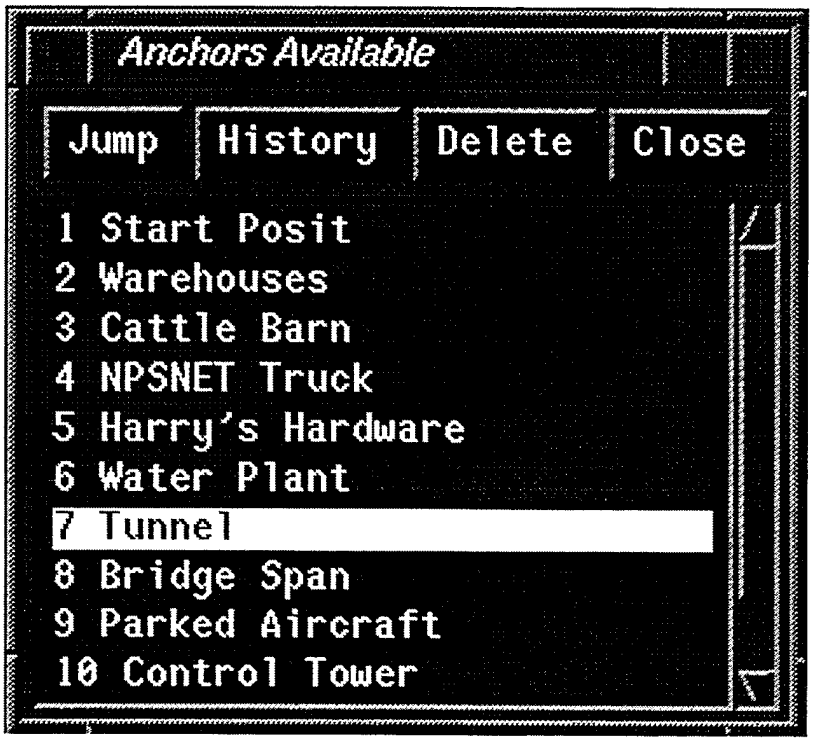

Figure 13: Interface Panel Anchors Available Window

position in NPSNET will be updated to zero. This feature was added to prevent those occasions where a moving vehicle jumps to an anchor that happens to be facing perhaps a wall, or possibly a cliff, thus sending the vehicle to its death. A jump can also be performed by double-clicking on an anchor name from within the list window. This will accomplish the same thing as using the **Jump** button.

3. History Button

The **History** button allows the user to go back to a previous anchor position, and becomes essential in large sized HyperSystems containing many anchors. Selection of the History button will cause the lister window to show the contents of the HyperStack, which reflects those anchors that have been visited. Additionally, the **History** button will now read **All Anchors**. The user can then take one of two possible actions. Either a past anchor can be traveled to, using the **Jump** button as described above, or the **All Anchors** button

can be selected to redisplay the complete anchor listing with no change in vehicle location. Selection of the **Jump** button will also redisplay the complete anchor listing, after performing the jump, and return the **History** button to the display.

4. Back Button

Another anchor traversal feature provided to the user is the ability to "back up" through the previous anchors visited. Unlike the previous functions described here, the **Back** button is not located in the Anchors Available window, since it does not relate directly to the selection of a specific anchor. Instead, it is placed along the bottom edge of the Interface Panel, along with the other Hypermedia Control Buttons, where it can be easily accessed without the need to bring up the Anchors Available window.

The **Back** button, when selected, simply inspects the current HyperStack and pops off the top anchor identifier, thus making the anchor below it the current anchor. The location of this new current anchor will be sent to NPSNET in the next outgoing communications socket, thus updating the vehicle position to that of the anchor. The Interface Panel will similarly be updated to reflect the new position and heading. The **Back** button can be selected as long as there are still previously visited anchors on the hyperStack. Once the stack has been emptied, the **Back** button will become disabled until future anchors are visited.

Deletion of anchors from the HyperSystem is accomplished using the **Delete** button within the Anchors Available window, and is described in detail in the section below covering anchor creation and editing.

D. ANCHOR AUTHORING AND EDITING

One of the great strengths of this hypermedia system is its ability to allow for dynamic authoring of anchors in the virtual world. As well as the previously discussed ability to delete anchors, facilities are also provided to create new anchors and modify existing ones "on the fly." This allows the user to modify his/her version of the virtual world, i.e., the current hypermedia file, and thus allow him/her to build up the training

database for future users. Currently, the anchor authoring capability is completely contained within the apparatus of the Interface Panel and is menu driven in nature. A future beneficial capability might allow the user to place anchors directly into the virtual world via a mouse and pointer scheme.

1. Building a New Anchor

As soon as a hyperfile has been opened, or a new file selected as current, the capability to create new anchors becomes enabled. To accomplish this action, the user must select the **New Anchors** option under the **Hypermedia-Edit** submenu. This will cause the Anchor Editor window to appear on top of the Interface Panel, and allow the user to begin creation of one or more anchors. The Anchor Editor is built from the ViewKit *VkPrefDialog* class, which provides the general structure for a preference dialog box. The programmer can then insert widgets as needed within the dialog window. Figure 14 shows an example of the Anchor Editor for modifying an existing anchor, which is identical in structure to that of the New Anchor Editor.

In the case of a new anchor, certain default information is provided to the user to assist in anchor definition. For example, assuming that the user will most likely drop an anchor at or near his/her current position in NPSNET, the position and orientation information in the New Anchor Editor default to the vehicle position and heading information taken from the most recent communication socket. The user can then define a meaningful anchor name, and an anchor type. At this time, the user also has the option of defining the information nodes to be attached to the anchor being defined. This is done by typing in the path and file name of the associated data file for a particular HyperNode. For example, the user may enter the path and name of a stored sound file into the "Audio Filename:" text entry window. It is not necessary that any attached data files be defined at anchor creation time, as an anchor can be later modified, and files added at that time.

When an anchor is created by selection of the *Ok* or *Apply* buttons in the Anchor Editor, the **EditAnchorWindow** class member function, *save()*, creates a new anchor and

four new nodes to be attached to it, and loads the window information into the appropriate anchor and node structures. This new anchor is then given a unique anchor identifier and attached to the end of the anchor linked list, and similarly to the bottom of the anchor storage table. Also at this time, the current Hypermedia file is flagged as needing to be saved, and the user will receive prompts to save the file as described earlier.

Upon defining a new anchor, the user can select the *Ok* button to accept the new anchor and close the editor dialog box. Alternately he/she can click on the *Apply* button to accept the new anchor, and then continue to define further new anchors by simply overwriting the information in each text entry window and selecting *Apply* again. Once this process is complete, selection of the *Ok* button will terminate the editor.

2. Editing an Existing Anchor

The user can also opt to modify an existing anchor in the HyperSystem, whether it be to update anchor positional information, or to change the attached files for an anchors's nodes. Figure 14 shows a sample Current Anchor Editor.

In order to modify an anchor, that anchor must be designated as the current one. The user can accomplish this by making a "Jump" to that anchor. Once an anchor in the HyperSystem has been traveled to, the **Current Anchor** option under the **Hypermedia-Edit** submenu will become enabled. Selection of this option will bring up the Anchor Editor dialog, loaded with the current anchor's data. As with the New Anchor Editor, information in the dialog window can now be modified by the user as desired. As soon as either the *Ok* or *Apply* button is selected, the new data will be saved to the HyperSystem, and will take effect for all future Panel and NPSNET interactions. If the *Cancel* button is selected prior to either of the other two, any modifications made will be ignored, and the editor dialog will terminate.

3. Deleting an Anchor

The **Delete** button on the Anchor Lister (previously described) is used to dynamically remove an anchor from the current HyperSystem, and involves more

Figure 14: ***Current Anchor Editor*** **Dialog Window**

complexity than meets the eye. Operationally, selection of **Delete** will remove the currently highlighted anchor from the listing, as well as within the NPSNET virtual world. To remove the anchor from the HyperSystem, the anchor linked list is updated to reflect a deletion, and then the anchor table must be updated. This involves decrementing the index of each of the subsequent anchors in the table by one, thus moving them "up" the table. Additionally, the node linked list and table must similarly be updated, but since each anchor contains facility for links to four nodes, each node in the table must be "moved up" four places.

To remove the deleted anchor from display in NPSNET, the communications socket to NPSNET is updated by the change in the HyperSystem so that, when the next socket is written, it will indicate to NPSNET that the anchor has been removed. Within NPSNET, that anchor model will be removed from the overall Performer anchor subtree and will no longer be drawn in the scene.

E. INFORMATION NODE ACCESS

There are a two primary ways in which an anchor's HyperNode information can be accessed while running the Interface Panel with NPSNET. One method has already been described in this report, and involves setting the Auto Anchor View function and then selecting whatever node types are desired to be automatically displayed.

The second, and more direct method for displaying node information is via the Hypermedia Buttons along the bottom edge of the Panel. Figure 15 below shows how these buttons appear on the Interface Panel.

Figure 15: Interface Panel Hypermedia Control Buttons

Among the six Hypermedia Control Buttons there are two categories, or types, of functions. The last two buttons are used to bring up the current HyperSystem Anchor Listing and to back up through the HyperStack, respectively, and have been previously detailed in the Anchor Navigation section. The first four buttons, however, all have to do with accessing an anchor's HyperNode information files.

When an anchor has been traveled to, it necessarily becomes the current anchor. Any information nodes attached to that anchor in the HyperSystem will be registered and the existing types will be reflected among the first four hypermedia buttons by enabling those that are appropriate. For example, in Figure 15 above, the current anchor has attached to it an audio node and a graphical image node. If any of the enabled buttons is selected, the associated database file will be displayed as appropriate. Audio files will be played over the local sound system, if one exists. Video files will be displayed using the SGI Movieplayer tool, which allows for multiple viewings of the clip, pause capability, and rewind, among other features. Graphical files will be displayed as images within a specially created ViewKit *VkSimpleWindow* object. Subsequent selection of this option will terminate display of the graphic window. And finally, text files are also displayed within a ViewKit *VkSimpleWindow* object, and are terminated in the same manner as the graphical window.

VI. CONCLUSIONS AND FUTURE WORK

A. RESULTS OF WORK

The result of this thesis work is that NPSNET-IV.8 now possesses a fully functioning hypermedia system. This system provides the user the capability to traverse the virtual world via the anchor system, as well as the ability to author new anchors and modify existing ones.

This hypermedia capability adds an entirely new facet to NPSNET, extending it into more of a training role. It provides the potential for users to have access to massive amounts of data with the simple click of a mouse, while easily traversing the world. This capability can be viewed from one of two standpoints. In a purely training role NPSNET can be used with the full hypermedia capability, enabling a single user, or a small number of locally networked users, to traverse a world via anchor jumps and access information nodes along the way. Conversely, in an auxiliary role, only the information access functions of the hypermedia system can be used, thus allowing for more realism as an entity traverses the world in the conventional manner. This way, the entity will behave as expected globally over a large network, yet still provide the benefit of information access locally to the user.

With the growth of the World Wide Web (WWW) and the Virtual Reality Modeling Language (VRML), a draft language specification for adding 3D data to the WWW, the extent to which hypermedia can be extended is only limited by one's imagination. It is not inconceivable to imagine a virtual simulation system possessing a hypermedia capability that provides direct access into the WWW, thus enabling the user to venture far beyond the confines of the defined virtual world.

Below, some of the key features of the NPSNET Hypermedia System are revisited. These include the user interface, and the authoring capabilities of the system.

1. User Interface

A consistent user interface is a significant feature of any window and menu driven application. This idea drove the development of the NPSNET hypermedia system interface features. Its pull-down menu and pop-up window mechanisms are certainly familiar to users of any modern GUI-based computer system, thus instilling a level of confidence in the mind of even the most inexperienced user.

The intelligence of an interface is also a key factor. The hypermedia interface makes extensive use of so-called "smart buttons" which automatically sensitize and desensitize based on previous selections and existing conditions. For example, the information node hypermedia buttons along the lower edge of the interface panel are enabled only when a hypermedia data file has been opened and an anchor possessing those information nodes is traversed to. Similarly, when closing an open data file, the system will automatically query the user as to whether or not he/she wants to save any changes made, if need be. These features of the interface provide a much more intuitive and comfortable environment for all levels of users.

2. Anchor Authoring Capability

The hypermedia system allows assignment of information nodes to each anchor. Additionally, these nodes can have attached to them data files of type audio, video, graphical image, or text, and can be assigned to specific anchors dynamically. Anchors and nodes can be added to either an open data file, or to a new data file under construction. This capability enables the authoring of both new anchors into an *existing world*, as well as *new worlds* themselves.

B. FUTURE WORK AREAS

This thesis lays a solid foundation for a hypermedia capability in NPSNET. However, there are areas for further expansion. Three areas are discussed below that highlight future research which could improve the overall capability of the system.

1. Interface Expansion

Currently, all interaction with the hypermedia anchor system is conducted through the Control-Interface Panel. While this provides all the functionality needed, it tends to take away from the user's feeling of "insertion" into the virtual world. A solution to this shortcoming would be to add a sophisticated user interface into the 3D world. This could include the ability for the user to author anchors by "mouse clicking" specific locations in the world, or modifying anchor location with a mouse dragging scheme.

The problem encountered in implementing the above lies in the way the interface is currently designed. There is no sense of interaction with the workstation running the NPSNET session, so there is no easy and direct method by which mouse inputs can be read and acted upon. This leads one to surmise, then, that the solution to the above idea will almost certainly involve redesigning the current input structure of the NPSNET and Interface Panel combination.

Another interface feature might be to display anchor icons within the radar display, and then allow the user the capability to use this interface as a means of "picking" anchors to jump to in the virtual world. While this in itself does not involve direct 3D world interaction, it does provide a 2D interface that might be more intuitive, and provide a bit more realism, than the current menu driven interface.

2. Variable Anchor Types

The Anchor class built into NPSNET maintains a *type* parameter, but in its current version has the capability to be of only one type. Anchors are assigned to a specific location in the virtual world, and are thus defined to be of type **TERRAIN**. An expansion on this would be to provide for varying anchor types within the Hyper System. Examples could include a vehicle anchor, which varies its location as the vehicle it is attached to moves through the world, or a temporal anchor, which exists for only a finite amount of time, and then deletes itself from the Hyper System at a preset time. These new anchor types could perhaps be differentiated by varying colors schemes, or anchor geometry.

As mentioned, the functionality for different anchor types already exists in the current Anchor class definition. The incorporation of code required to actually display varying types would be straightforward. Adding the functionality for these different anchor types would be more difficult, and would primarily involve redefining the anchor objects to act in a new manner, as mentioned in the above examples.

3. Hypermedia Datafile Access

While anchor authoring is fully functional in the current hypermedia system, attachment of data files to associated information nodes is somewhat cumbersome. The user must know in advance where the desired data file exists in the directory structure, and type this path into the appropriate dialog window. A better means of authoring would be to allow the user to somehow preview data files during the authoring process, so that he/she could view potential files, and then accept one from among these.

Also along these lines, a means might be provided to let the user view existing node data files by type, or possibly to view particular anchors by what node type they possess. To illustrate an example, a user could be allowed to access all those anchors that have any video node attached to them, or to see a listing of those anchors that have an audio node containing a specific data files.

APPENDIX: NPSNET HYPERMEDIA USER'S MANUAL

Included here is a user's manual covering use of the newly incorporated hypermedia capabilities of NPSNET-IV.8.

A. STARTING NPSNET AND THE INTERFACE PANEL

In order to run NPSNET under the control of the interface panel, first start an NPSNET-IV.8 session from within the ~npsnetIV/npsnetIV.8/bin directory as you normally would, but with the inclusion of the '-r' switch, for example:

```
unix prompt>: npsnetIV -r <other switches>
```

To run the interface panel on a different workstation, enter the command 'panel' from the ~npsnetIV/npsnetIV.8/bin directory, using the '-s' switch to designate the local NPSNET session host. For example, if NPSNET is being run on a local workstation called meatloaf, the interface panel would be started by entering:

```
unix prompt>: panel -s meatloaf <other switches>
```

Note: In order to see a listing of other possible switches for the interface panel, simply enter the '-h' switch for help, i.e., 'panel -h'.

Finally, once the NPSNET session has come up, hit the 'i' key on the keyboard until the display shows the input source as being **RMT**, for remote panel.

B. SELECTING A VEHICLE TYPE

Once the interface panel has come up, inputs from the NPSNET session should be visible on the various panel displays (assuming NPSNET is also up and running). The first step is to select on the panel the type of vehicle being run in NPSNET. To do this, select the **Vehicle** pull-down menu. From there, select one of either **Stealth**, **Aircraft**, **Tank**, or **ASW Helo**. Immediately, the appropriate display for the vehicle type should appear on the interface panel.

C. OPENING A HYPERMEDIA DATABASE FILE

Before beginning a hypermedia session in NPSNET, either a new database file or an existing one must be opened. To do this, select the **Hypermedia** pull-down menu, and under that the **File** submenu. Under the **File** submenu will appear the options to open an existing file or start a new one (all other options will be disabled at this point). If a new file is desired, simply select **New**. To open a file, select the **Open** option, and then use the resulting file access dialog box to open a database file.

D. SETTING HYPERMEDIA PREFERENCES

There are a variety of ways by which a user may set interface panel preferences. Under the **Hypermedia** pull-down menu, selection of the **Display** submenu will result in a menu showing two option selectors, **Anchors On** and **All Anchors**. The first indicates that hypermedia anchors are to be displayed in the NPSNET virtual world. If this option is selected, the selector will change to **Anchors Off**, and will indicate that anchors will not be displayed. The second option selector indicates that all of the anchors in the hypermedia file database are to be displayed. By selecting this option, the selector changes to **Local Anchors Only**, and indicates that only those anchors within a specified range of the vehicle will be displayed in the virtual world.

The other way to set user preferences for the interface panel is through the Preferences window, which can be displayed by selecting the **Hypermedia** pull-down menu, then the **Edit** submenu under that, and finally the **Preferences** selector. The Preferences window gives the user the option of setting the Auto Anchors and Local Anchors functions of the hypermedia system. By selecting the **Auto Anchors** toggle, travel within a specified range of a hypermedia anchor in the virtual world will cause its information nodes to be displayed. Exactly which information nodes display is set in this window by selecting the **Audio**, **Video**, **Graphic**, and **Text** toggle buttons. The range out to which the Auto Anchors function activates these nodes is set in the range window under the **Auto Anchors** toggle by simply entering the desired number in the dialog box.

Also in the Preferences window is a **Local Anchors Only** toggle button, which is simply another place where this function can be selected. Selection of this toggle will update the **Option** menu selector discussed earlier, and vice versa. More importantly, though, this window enables the user to enter the range out to which local anchors will be displayed in the virtual world, again by simply entering the desired number in the dialog box.

E. EDITING HYPERMEDIA ANCHORS

Once a hypermedia data file has been opened, or a new one started, anchors can be added or modified through the Anchor Editor window. Depending whether a new anchor is to be added, or an existing one modified, the editor is accessed by selection of the **New Anchor** or **Current Anchor** selector, as appropriate. Both of these selectors are found under the **Edit** submenu, which is under the **Hypermedia** pull-down menu.

Selection of **New Anchor** will open the anchor editor with default values in each of the data entry locations. From here, anchor data to include name, type, orientation, location, and data node filenames can be entered into each of the appropriate dialog boxes. Selecting the **Ok** button will cause the new anchor to be accepted, and will close the window. Selecting **Apply** will cause the new anchor to enter, but leave the window open, thus allowing for further anchors to be added. Finally, the **Cancel** button will cancel any entries made subsequent to hitting the **Apply** button, and close the window.

To modify the stored parameters for an existing anchor, the desired anchor must first be made the current one. This is done by jumping to that anchor (to be discussed later). Once this anchor is the current, select the **Current Anchor** selector, as described above. The anchor editor window will now appear with the data for that anchor, which can then be modified. The **Ok**, **Apply**, and **Cancel** buttons work the same as above.

F. ANCHOR TRAVERSAL AND NODE SELECTION

Once a hypermedia database has been opened and preferences set, the virtual world can then be traversed via the anchor system, and information nodes accessed. The primary

functionality for using the hypermedia system is located along the bottom edge of the interface panel.

To access the anchors, select the **Anchors Available** button. This will display a window showing a listing of all the anchors, as well as buttons to manipulate them. To get to a specific anchor location, either double click on that anchor's name in the list, or select the name and click on the **Jump** button. As anchors are traversed, a stack is maintained of visited anchors. To return to a previous location, select the **History** button. This will change the anchor list window to show the stack. Simply jump back to the desired anchor, or hit the **All Anchors** button to bring back the normal anchor listing.

Once an anchor has been jumped to, and it has thus been designated as current, the information node buttons along the bottom of the panel will become enabled as appropriate. Specifically, whatever types of information are attached to the current anchor will cause the corresponding information buttons to become sensitive. Simply selecting the desired button - **Audio**, **Video**, **Graphic**, or **Text** - will cause either a sound clip, a video player image, a graphical image, or a text box to appear in the interface panel area.

G. SAVING A FILE AND EXITING THE PROGRAM

In order to save an opened hypermedia file select the **Hypermedia** pull-down menu, then the **File** submenu. Under this submenu are two selectors for **Save** and **Save As** functions. The **Save** button will only be enabled if a previously opened file is currently open, and will simply update the stored version. If a new file was opened, then the **Save As** button must be used to save the file. Its selection will bring up a Save File dialog box, which will allow the user to enter a path and file name under which to store the current hypermedia database.

To exit the interface panel program, select the **Application** pull-down menu, and then the **Quit** selector button. A dialog box will appear to double check the desire to quit. If a current hypermedia file is open, and changes have been made to it, the user will be

prompted to save the file. Whether saved or not, the current file will be closed before the program is exited.

LIST OF REFERENCES

[BUSH45] Bush, Vannevar., *As We May Think*, The Atlantic Monthly, July 1945.

[CARD91] Card, Stuart K., Robertson, George G., and Mackinlay, Jock D., "The Information Visualizer: An Information Workspace," *Human Factors in Computing Systems*, ACM SIGCHI Conference Proceedings, 1991.

[COOK88] Cook, Peter., *An Encyclopedia Publisher's Perspective*, Interactive Multimedia, Apple Computer Inc., Microsoft Press, 1988.

[LOMB93] Lombardo, Charles P., "Hyper-NPSNET: Embedded Multimedia in a 3D Virtual World", Master's Thesis, Naval Postgraduate School, Monterey, California, September 1993.

[MCMA94] McMahan, Christopher B., "NPSNET IV: An Object-Oriented Interface for a Three-Dimensional Virtual World, Master's Thesis", Naval Postgraduate School, Monterey, California, December 1994.

[NELS65] Nelson, Ted., *A File Structure for the Complex, The Changing and the Indeterminate*, ACM 20th National Conference, 1965.

[NELS80] Nelson, Ted., "Replacing the Printed Word: A Complete Literary System", *Information Processing '80*, 1980.

[NELS87] Nelson, Ted., "All For One and One For All", *Hypertext '87 Proceedings*, November 1987.

[NIEL90] Nielsen, Jakob., *Hypertext and Hypermedia*, Academic Press, 1990.

[PRAT94] Pratt, David R., Zyda, Michael J., and Kelleher, Kristen M., "1994 Annual Report for the NPSNET Research Group", Naval Postgraduate School, Monterey, California, 1994.

[RAO90] Rao, Usha, and Turoff, Murray., "Hypertext Functionality: A Theoretical Framework", *International Journal of Human-Computer Interaction*, 1990.

[RAYM88] Raymond, Darrell R., and Tompa, Frank W., "Hypertext and the Oxford English Dictionary", *Communications of the ACM*, July 1988.

[SERB94] Serbest, Fikret., "An Automated Tactical Operations Command, Control, Communications, and Intelligence Planning Tool using Hyper-NPSNET", Master's Thesis, Naval Postgraduate School, Monterey, California, September 1994.

[SMIT88] Smith, John, and Weiss, Stephen F., "An Overview of Hypertext", *Communications of the ACM*, July 1988.

[VIEW94] "Iris ViewKit Programmer's Guide," Online Reference Manual available with the Silicon Graphics Irix 5.3 Operating System.

[ZYDA93] Zyda, Michael J., Pratt, David R., Falby, John S., Barham, Paul T., and Kelleher, Kristen M., "NPSNET and the Naval Postgraduate School Graphics and Video Laboratory", *Presence*, December 1993.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
Dudley Knox Library Code 013 Naval Postgraduate School Monterey, CA 93943-5101	2
Chairman, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943	2
Professor David R. Pratt, Code CS/Pr Computer Science Department Naval Postgraduate School Monterey, CA 93943	2
Mr John S. Falby, Code CS/Fa Computer Science Department Naval Postgraduate School Monterey, CA 93943	2
Dr Michael J. Zyda, Code CS/Zk Computer Science Department Naval Postgraduate School Monterey, CA 93943	10
Mr Paul Barham, Code CS/Barham Computer Science Department Naval Postgraduate School Monterey, CA 93943	1
Dr Donald Brutzman, Code UW/Br Naval Postgraduate School Monterey, CA 93943	1
Lt Alan B. Shaffer 618 Galen Drive San Jose, CA 95123	2